A3P 2014/2015
Groupe 1

I.A) Auteur

Victor Bouilleux

I.B) Thème

Sur une piste, un jeune médecin doit arriver en bas avant l'avalanche

I.C) Résumé du scénario

Un homme est en haut d'une piste de ski. C'est alors que se déclenche une avalanche derrière lui. Les remontées mécaniques étant hors d'usage, il est obligé de passer par la piste pour arriver en bas. Problème : la piste est parsemée d'obstacles.

Premier obstacle : plaque de verglas

Deuxième obstacle : champ de bosses

Troisième obstacle : barre rocheuse

Quatrième obstacle : crevasse au milieu de la piste

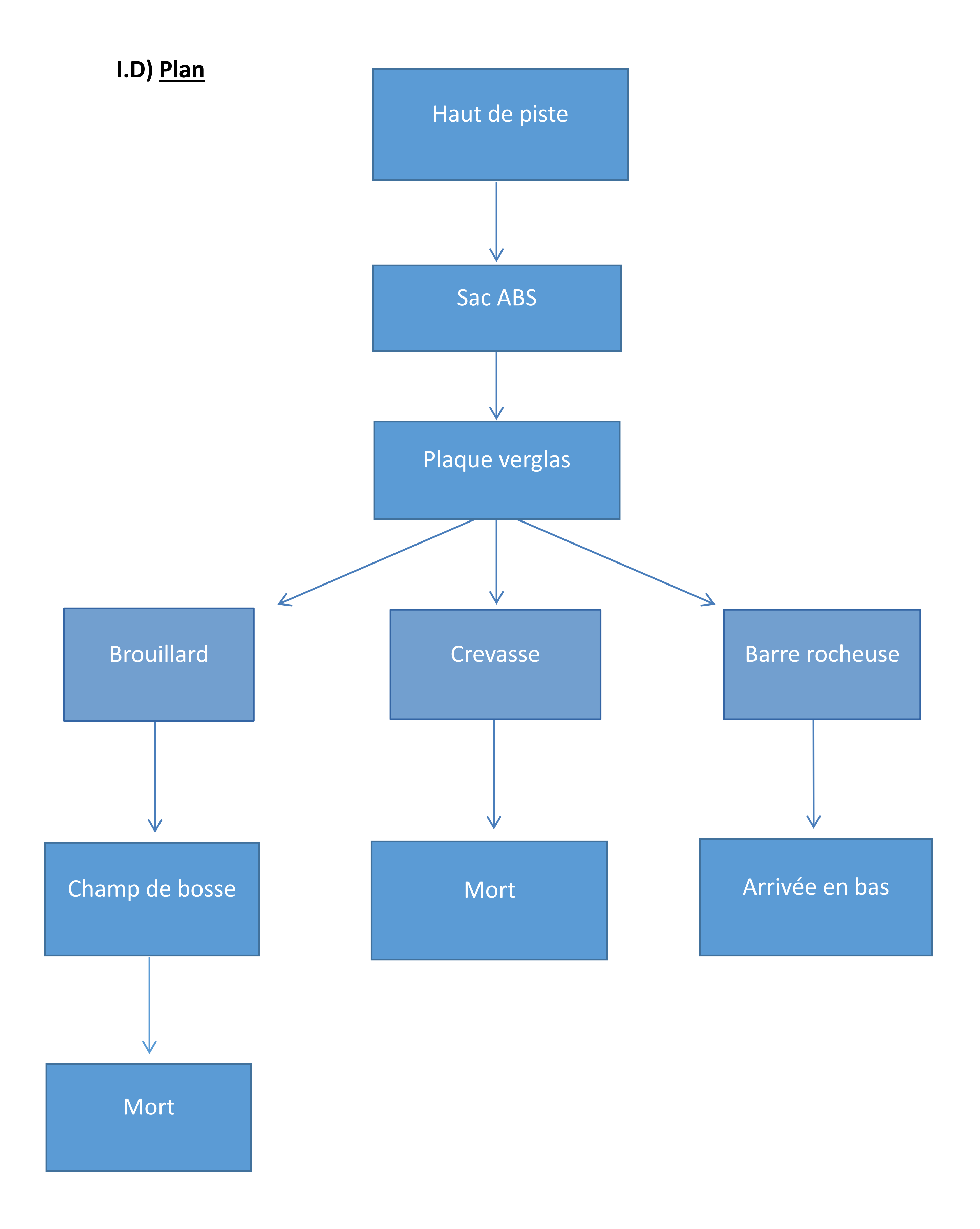
Cinquième obstacle : plaque de brouillard

Sixième obstacle : tronc arbre sur la piste

Si l'homme parvient à passer tous les obstacles sain et sauf, il arrive en bas de la piste et gagne le jeu. En revanche, s'il chute sur un obstacle il se fait emporter par l'avalanche te meurt.

Dès le début du jeu, le joueur part avec un « bonus vie » qu'il pourra utiliser s'il le souhaite dans le jeu pour ne pas mourir.

I.D) Plan



Haut de piste
Sac ABS
Plaque verglas
Brouillard
Crevasse
Barre rocheuse
Champ de bosse
Mort
Arrivée en bas
Mort

I.E) <u>Scénario détaillé</u>

Un homme est en haut d'une piste de ski. C'est alors que se déclenche une avalanche derrière lui. Les remontées mécaniques étant hors d'usage, il est obligé de passer par la piste pour arriver en bas. Problème : la piste est parsemée d'obstacles.
Il devra obligatoirement passer tous les obstacles pour pouvoir arriver en bas sain et sauf. A chaque obstacle, le joueur aura deux choix possible : passer l'obstacle et continuer ou alors ne pas le passer. Dans ce cas, il se fera emporter par l'avalanche et mourra.

Le joueur démarre en haut de la piste. L'avalanche se déclenche. Le skieur part tout droit et arrive sur le premier obstacle, une plaque de verglas. S'il passe ce premier obstacle, trois choix s'offrent à lui : continuer tout droit, partir à gauche ou partir à droite.

S'il part à gauche, il devra passer plusieurs obstacles : d'abord une plaque de verglas, puis un champ de bosse. S'il passe ces deux obstacles, il arrive en bas de la piste avant l'avalanche et gagne le jeu.

S'il part tout droit, il chute dans une crevasse et meurt. C'est donc un « piège ».

S'il part à droite, il tente de franchir une barre rocheuse mais tombe et se fait emporter par
S'il l'utilise, il ne se fera pas emporter par l'avalanche et arrivera en bas de la piste sain et sauf. Dans le cas où le joueur n'utilise pas le bonus, il se fera emporter par l'avalanche et mourra.

I.F) <u>Détails des lieux, items, personnages</u>

Mon jeu est composé de 11 « pièces » différentes et avec 6 lieux différents. Le personnage reste le même durant toute la durée du jeu.

I.G) <u>Situations gagnantes et perdantes</u>

Le joueur gagne le jeu s'il parvient à arriver en bas de la piste avant l'avalanche. Si l'avalanche l'emporte, il meurt et donc perd.

I.H) <u>Commentaires</u>

Je me suis arrêté au 7.40 dans la liste officielle des exercices par manque de temps pour finir le projet. Mis à part une commande qui ne fonctionne pas, le jeu fonctionne parfaitement.

II) Réponses aux exercices

- 7.4

Modification de la méthode createRooms() de la classe Game

- 7.5

Création d'une procédure printLocationInfo() dans la classe Game, reprenant le code redondant des méthodes createRooms() et goRoom().
Insertion de la procédure printLocationInfo(),

- 7.6

Ajout de l'accesseur getExit() dans la classe Room
Utilisation de la classe getExit() dans la classe Game

- 7.7

Modification de printLocationInfo()
Définition de getExitString()

- 7.8

Ajout de la HashMap + ajout d'un déplacement vertical

- 7.9

Cette méthode keySet() permet d'accéder à la HashMap

- 7.10

Création de la méthode getExitString()

- 7.11

Création de la méthode getLongDescription() dans la classe Room() et modification de printLocationInfo()

- 7.14

Création de la commande look dans la classe CommandWord, création de la méthode look dans la classe Game et création de else if dans la classe Game

- 7.15

Création de la commande eat dans la classe CommandWord et création de la méthode eat()

- 7.16

Création de showAll et de showCommands

- 7.18

Création de getCommandsList, comparaison des classes CommandWords, Parser et Game avec le zuul-better et le zuul-with-images et ajout d'un bouton

- 7.19

Déplacement des images du jeu à la racine du dossier et changement du nom des chemins d'accès des images

- 7.20

Création d'une classe Item qui contient deux attributs. Le premier concerne le poids de l'objet et le second sa description. On ajoute ensuite ces objets à la classe Room par le biais d'une HashMap.

- 7.21

Ajout de deux méthodes : itemDescription qui permet la modification de la description si il y a des items et getItemString qui permet l'affichage des items d'une Room les uns à la suite des autres.

- 7.22

Recours à la HashMap afin d’associer le nom de l’item à l’objet en question

- 7.22.2

Ajout de l’item dans le jeu

- 7.23

Réalisation de la commande back qui permet au joueur de revenir dans la pièce précédente. Pour cela, on crée la méthode back dans la classe GameEngine

- 7.26

Utilisation de l’attribut stack afin de créer une commande permettant de revenir au début du jeu.

- 7.26.1

Création d’un dossier contenant les deux javadoc contenant les lignes :
- javadoc -d progdoc -author -version -private -linksource
- javadoc -d userdoc -author -version

- 7.28.1

Création d’une commande test qui teste le chemin le plus court pour arriver à la fin du jeu. Pour cela, il faut créer un fichier dans lequel on rentre les commandes permettant d’accéder le plus rapidement possible à l’arrivée.

- 7.29

Création de la classe Player qui a pour but de concentrer toutes les commandes auxquelles le joueur aura accès durant le jeu. Pour cela, création des attributs Name pour le nom du joueur, WeightMax pour le poids maximal d’objets que le joueur peut porter sur lui, Weight pour le poids d’objet que le joueur porte sur lui et CurrentRoom pour la pièce dans laquelle se trouve le joueur.

- 7.30

Nouvelles méthodes take et drop qui permet de prendre des objets trouvés dans des pièces et de les mettre dans l’inventaire ainsi que de relâcher ces objets au cours du jeu.

- 7.31

Création de la classe Itemlist qui répertorie les différents objets trouvés. C’est grâce à cette classe que le joueur pourra jeter un objet ou alors vérifier le contenu de son inventaire.

- 7.32

Méthode par laquelle on détermine le poids maximum que le joueur pourra porter sur lui, ce poids étant calculé avec les objets trouvés dans les différentes pièces du jeu.

- 7.33

Cette commande inventaire permet au joueur de voir les différents objets qui se trouvent sur lui.

- 7.34

Exercice non traité

- 7.35

Création d’une classe CommandWords «enum » qui répertorie tous les noms des commandes que l’on peut utiliser dans le jeu.

- 7.35.1

Le switch permet de reconnaitre les commandes tapées dans la classe CommandWords « enum ».

- 7.40 – 7.41

Création de deux nouvelles commandes help et look. La première permet au joueur de lui remémorer les différentes possibilités qu'il a en fonction de la pièce où il se trouve et la seconde commande permet au joueur de regarder s'il y a des objets dans la pièce où il se trouve.

III) Mode d'emploi

Voici les différentes commandes qui permettent de jouer :

- Go : permet au joueur de se délacer
- Back : permet de retourner dans la pièce précédente
- Look : permet de regarder s'il y a des objets dans la pièce où le joueur se trouve
- Test : permet au joueur de prendre le chemin le plus court pour arriver à la fin du jeu
- Take : permet au joueur de prendre un objet dans une pièce
- Drop : permet au joueur de relâcher l'objet qu'il a pris

IV) Déclaration anti-plagiat

Afin de réaliser ce projet, j'ai demandé de l'aide à Alexandre Béchir pour la partie code du projet